Implicit integration methods for dislocation dynamics

D. J. Gardner, C. S. Woodward, D. R. Reynolds, G. Hommes, S. Aubry, A. Arsenlis

July 1, 2014

Modelling and Simulation in Materials Science and Engineering

Implicit integration methods for dislocation dynamics

D J Gardner[1], C S Woodward[1], D R Reynolds[2], G Hommes[1], S Aubry[1], and A Arsenlis[1]

[1]Lawrence Livermore National Laboratory, Livermore, CA 94551, USA

[2]Department of Mathematics, Southern Methodist University, Dallas, TX 75275, USA

E-mail: `gardner48@llnl.gov`, `woodward6@llnl.gov`, `reynolds@smu.edu`, `sylvie.aubry@llnl.gov`, `arsenlis@llnl.gov`

Abstract. In dislocation dynamics simulations, strain hardening simulations require integrating stiff systems of ordinary differential equations in time with expensive force calculations, discontinuous topological events, and rapidly changing problem size. Current solvers in use often result in small time steps and long simulation times. Faster solvers may help dislocation dynamics simulations accumulate plastic strains at strain rates comparable to experimental observations. This paper investigates the viability of high order implicit time integrators and robust nonlinear solvers to reduce simulation run times while maintaining the accuracy of the computed solution. In particular, implicit Runge-Kutta time integrators are explored as a way of providing greater accuracy over a larger time step than is typically done with the standard second-order trapezoidal method. In addition, both accelerated fixed point and Newton's method are investigated to provide fast and effective solves for the nonlinear systems that must be resolved within each time step. Results show that integrators of third order are the most effective, while accelerated fixed point and Newton's method both improve solver performance over the standard fixed point method used for the solution of the nonlinear systems.

Submitted to: *Modelling Simulation Mater. Sci. Eng.*

1. Introduction

Dislocation dynamics (DD) is a mesoscale method connecting the physics of dislocations with the evolution of strength and strain hardening in crystalline materials. It simulates explicitly the motion, multiplication, and interactions of discrete dislocation lines, the carriers of plasticity in crystalline materials, in response to an applied load [1]. The challenge in connecting the aggregate behavior of dislocations to macroscopic material response has been one of computability. In DD, one needs to be able to trace the simultaneous evolution of millions of dislocation lines over extended time intervals, in order to directly compute the plastic strength of crystalline materials [2].

The system composed of moving and interacting dislocation segments is stiff. Two aspects may affect the motion of dislocation nodes: the motion in response to mechanical forces, and the motion in response to mesh optimization. An analysis of the eigenvalues of the Jacobian matrix for a simple dislocation configuration shows that some eigenvalues are large and others are small. In other words, some component(s) of the dislocation node positions oscillate with high frequencies while others do not. For this class of problems, an implicit numerical integrator needs to be used, as demonstrated in the recent paper by Sills and Cai [3].

In this paper we are concerned with accurate and efficient numerical time integrators to obtain dislocation segment displacements from velocities calculated at the nodes. However, unlike the numerical integration methods in [3], we focus only on provably accurate and stable methods utilizing fully implicit algorithms. Each time step in a fully implicit simulation includes a nonlinear iteration to solve for new dislocation node locations. Each of these iterations in turn requires nodal force calculations, and these force calculations account for about 90% of run time. As a result, efficient nonlinear solvers and time integrators are essential for effective simulations.

In dislocation dynamics codes, the standard methods of time-integration are the forward Euler and the trapezoid algorithms [4, 5, 6, 7, 8]. The optimal parameters for these algorithms are studied in detail in [3, 8] for a Frank-Read source. Other time integration schemes have not been systematically studied in the literature.

A standard integrator in use in many application areas is the trapezoidal method [9]. This method provides second order accuracy while only requiring information from the last prior time step. As such, it is the simplest implicit second order method to implement, and this was the original integration method implemented for the implicit formulation in `ParaDiS` [1]. Higher order time integrators have the potential to provide either greater accuracy for a given time step size or the same accuracy with a larger time step. In the case of DD simulations, finding balance between good accuracy and larger time steps is desired.

Furthermore, due to the adaptive nature of the models of the dislocation discretization and discontinuous topological events, use of many prior time step solutions is problematic. For this reason, linear multistep methods [9, 10], where multiple previous solutions are required to achieve second or higher orders of accuracy, are not well-suited for DD. Instead, use of multi-stage methods [9, 10], where only one prior solution is required to achieve higher orders, can be a better fit. However, multi-stage methods require the solution of multiple nonlinear problems, or a single but larger nonlinear problem, within each time step. Hence a careful implementation of these methods, along with the necessary nonlinear solvers, is essential for effective performance.

With larger time steps, the underlying nonlinear problems that must be solved for each time step solution can be more challenging, due to the increased nonlinearity from the larger time span. The easiest nonlinear solver to implement is a fixed point iteration [11]. While this method can be very effective, its linear rate of convergence can lead to long run times. Alternatively, Newton's method with its quadratic rate of convergence can be much faster. However, Newton's method requires both a good initial iterate and an effective linear solver to be applied within each Newton iteration. In recent years, an acceleration method for fixed point iterations has been used effectively in various applications to speed up traditional fixed point solves, making the accelerated method competitive with Newton's method [12, 13, 14, 15].

This paper develops and demonstrates a new and effective solution strategy for implicit formulations of DD. The strategy uses Diagonally Implicit Runge-Kutta time integrators and Anderson accelerated fixed point solvers for the nonlinear systems. Results show significant speedup over a trapezoid integrator with a fixed point nonlinear solver. The rest of the paper is organized as follows. Section 2 presents the DD model and the differential system being integrated. Section 3 presents the integrators developed and compared in this study, and Section 4 presents the nonlinear solvers. The software used for all numerical results given in Section 6 is described in Section 5. Lastly, conclusions on the benefits and limitations of the methods discussed in the paper are given in Section 7.

2. Mathematical Model

The interaction and evolution of dislocations constitute the dislocation dynamics method developed in `ParaDiS`. Dislocation lines are discretized as segments bounded by nodes. Several millions of segments can be present during a typical DD simulation and several hundred thousand timesteps are usually made to reach one percent of plastic strain, as a result DD simulations are computationally expensive and challenging.

Consider a dislocation segment $[x_i,\, x_j]$, composed of two nodes x_i and x_j. The total elastic force $F_i(t)$ at the node x_i is composed of (a) the force the segment exerts on itself, (b) the force associated with the dislocation's core structure, (c) the applied force the simulation domain is subjected to, and (d) the interaction force all the other segments in the simulation exert on the the dislocation segment $[x_i,\, x_j]$. This last force, the interaction force, is the most expensive calculation of a dislocation dynamics method as it scales as $\mathcal{O}(\mathcal{N}^2)$, where $\mathcal{N}$ is the number of dislocation segments that are local to the segment $[x_i,\, x_j]$. We note that a key component of `ParaDiS` allowing for scalable dislocation dynamics simulations is its use of a *Fast Multipole Method* [16] for efficient approximation of $F_i(t)$; however these force calculations are still the dominant cost in `ParaDiS` simulations.

Once the total force is determined, the velocity is calculated through the definition of a mobility law M, a constitutive property that characterizes the material being simulated. The velocity of node i is given as $v_i(t)$ by

$$v_i(t) = M(F_i(t)). \tag{1}$$

The mobility law M can be linear or non-linear depending on the the information known about the material. Examples of linear mobility laws are given in [17] and a non-linear mobility law in [18] for tantalum materials.

In DD simulations, the computational unknowns x_i correspond to positions of the nodes which change as a function of time using the standard equation of motion,

$$\frac{dx_i(t)}{dt} = v_i(t). \tag{2}$$

Two common practices used by DD simulators lead to potentially discontinuous behavior and can have direct impact on the choice of time integration method. The first arises from the piecewise linear discretization scheme for the dislocation segments described above. Here, the dislocation network is approximated by a set of lines whose positions are continuous, but whose tangents are discontinuous and whose curvature is undefined. Due to the inherent interaction between space and time in the equations of motion for each node, it is likely that this low degree of smoothness in the spatial approximation results in a correspondingly low degree of smoothness in the temporal dynamics of the dislocation network.

Second, a key component of the dislocation simulator, `ParaDiS`, is its ability to dynamically adapt the dislocation topology between time steps, allowing for the creation/deletion of new dislocation nodes and segments as the simulation proceeds. In typical simulations of crystalline dislocation dynamics, the number of dislocation nodes may vary by multiple orders of magnitude, especially in studies of strain hardening processes wherein the dislocation density increases dramatically from an initially simple state. This adaptivity process only occurs between time steps (not within steps), but is performed repeatedly throughout the course of a simulation.

3. Time Integrators

We consider two methods for the implicit time integration used for evolving the DD system (2). To define these, first consider the initial value problem (IVP) in general form

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0, \tag{3}$$

where t is the independent variable (time), $y(t) = [x_0, x_1, \ldots]$ contains all of the dependent state variables at time t, t_0 is the initial time, y_0 is the vector of initial conditions, and $f(t, y) = [v_0, v_1, \ldots]$ contains the right-hand sides for each dependent variable. In performing time integration with both methods, we denote y_n as our approximation to the solution $y(t_n)$, where we consider time steps $t_0 < t_1 < \ldots$, with step sizes $h_n = t_{n+1} - t_n$.

3.1. Trapezoid Integrator

The most common integrator used for implicit dislocation dynamics is the trapezoid integration method, where the solution at each new time solves the equation

$$y_{n+1} = y_n + \frac{h_n}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right). \tag{4}$$

Assuming sufficient continuity of f as a function of both y and t, this method is second order, meaning that for each time step the local error satisfies

$$\|y(t_{n+1}) - y_{n+1}\| \leq C h_{max}^3, \tag{5}$$

under the assumption that $y_n = y(t_n)$, where h_{max} is the maximum time step used over all steps to get from t_0 to t_n, and C is a constant independent of the time step sizes [9]. Globally, this method is $\mathcal{O}(h^2)$, due to accumulation of local errors from one step to the next. Thus, the accuracy of the approximate solution improves quadratically as the maximum time step size decreases.

The trapezoid method, also known as the Adams-Moulton 1-step implicit method, is the simplest second order 1-step method for implicit integration. Higher order implicit multistep methods, such as other Adams-Moulton or BDF methods, require use of multiple saved prior time step solutions. Given the adaptive nature of DD computations, these older solutions would need to be projected onto the current solution space at every update, and the resulting expense of these updates could become large. A 1-step method is advantageous within a DD setting since it does not require saving multiple old values.

Implementation of the method requires the solution of a nonlinear system of algebraic equations for the new time solution, y_{n+1}, within each time step. This system is defined as solving the nonlinear residual equation for y,

$$g(y) = y - y_n - \frac{h_n}{2}\left(f(t_n, y_n) + f(t_{n+1}, y)\right) = 0, \tag{6}$$

or more specifically through finding y such that

$$\|g(y)\|_\infty \leq \epsilon_n, \tag{7}$$

where ϵ_n is the nonlinear solver tolerance, and

$$\|g(y)\|_\infty \equiv \max_{1\le i\le N} |g_i(y)|$$

is the maximum norm.

Methods for such solves are discussed in Section 4. These solves can be accelerated through computation of an explicit predictor. At the first time step and for newly created nodes, `ParaDiS` uses the simple forward Euler predictor,

$$y_{n+1} = y_n + h_n f(t_n, y_n). \tag{8}$$

While this predictor has second order local error, it is also a 1-step method and thus does not require old solutions. For nodes where the value of f at the previous step is known, `ParaDiS` uses an explicit linear multistep predictor,

$$y_{n+1} = y_n + \frac{h_n}{2}\left(f(t_n, y_n) + f(t_{n-1}, y_{n-1})\right). \tag{9}$$

This predictor also has second order local error and requires saving the result of an old function evaluation. The value of y_{n+1} computed from these predictors is then used as the initial guess in the iterative nonlinear solver for the implicit solution.

Lastly, we note that `ParaDiS` updates the time step size based on the success or failure of the nonlinear solver, through either increasing the step size on a successful nonlinear solve, or decreasing the step size and retrying the step on a failed solve.

3.2. DIRK Integrator

We compare the above trapezoid integrator with higher-order embedded diagonally-implicit Runge-Kutta (DIRK) time integration methods, of the form

$$\begin{aligned}
z_i &= y_n + h_n \sum_{j=1}^{i} A_{i,j}\, f(t_n + c_j h_n, z_j), \quad i = 1, \ldots, s, \\
y_{n+1} &= y_n + h_n \sum_{j=1}^{s} b_j\, f(t_n + c_j h_n, z_j), \\
\tilde{y}_{n+1} &= y_n + h_n \sum_{j=1}^{s} \tilde{b}_j\, f(t_n + c_j h_n, z_j).
\end{aligned} \tag{10}$$

Here, the z_i denote the s internal stages of the Runge-Kutta method, each of which explicitly depend on preceding stages and implicitly depend on only themselves. The new time solution is given in y_{n+1}, and an *embedded* solution is given in $\tilde{y}_{n+1}$. Each method is uniquely defined through the coefficients $A_{i,j}$, c_i, b_j and $\tilde{b}_j$, $i, j = 1, \ldots, s$, where DIRK methods are characterized through the condition that $A_{i,j} = 0$ for $j > i$. We note that unlike many standard Runge-Kutta methods, embedded RK methods naturally allow for time step adaptivity based on a reliable estimate of the solution error, since both y_{n+1} and $\tilde{y}_{n+1}$ give different approximations for $y(t_{n+1})$. Moreover, embeddings result in significantly less expensive error estimates than those resulting from either Richardson extrapolation or deferred-correction techniques.

In order to implement the method (10), in each time step we sequentially solve s implicit systems for the stage solutions z_i. To this end, at each stage we define the nonlinear residual equation

$$g(z) = z - h_n\, A_{i,i}\, f(t_n + c_i h_n, z) - y_n - h_n \sum_{j=1}^{i} A_{i,j}\, f(t_n + c_j h_n, z_j) = 0, \quad (11)$$

and we compute z_i as the solution to this nonlinear algebraic system of equations $g(z) = 0$. Our algorithms for solving these systems are shared with the trapezoid method from Section 3.1, and will be discussed in detail in Section 4.

We accelerate convergence of the nonlinear solvers through supplying an explicit predictor of the stage solution, $z_i^{(0)}$. In the computed results (Section 6), we investigate three approaches for constructing the initial guess for each stage over the time step $t_n \to t_{n+1}$:

(a) use the solution from the beginning of the step, $z_i^{(0)} = y_n$,

(b) use the previous step solution for the first stage, $z_1^{(0)} = y_n$, followed by use of the previous stage solution for subsequent steps, $z_i^{(0)} = z_{i-1}$,

(c) use the previous step solution for the first stage, $z_1^{(0)} = y_n$, and the quadratic Hermite interpolant through $\{y_n, f(t_n, y_n), f(t_n + c_k h_n, z_k)\}$ for the remaining stages,
$$z_i^{(0)} = y_n + f(t_n, y_n)\left(\tau - \frac{\tau^2}{2h_n c_k}\right) + f(t_n + c_k h_n, z_k)\frac{\tau^2}{2h_n c_k}$$
where $\tau = h_n c_i$ [19]; we choose c_k so that $c_k \neq 0$ and $c_k > c_j$ for all $j = 1, \ldots, i-1$.

In addition to allowing increased accuracy and stability over linear multistep methods (e.g. trapezoid), embedded DIRK methods allow efficient estimates of temporal error, thereby allowing for robust step size adaptivity methods. To this end, we compute the temporal error estimate

$$e_n = \|y_n - \tilde{y}_n\|_{WRMS} = \left(\frac{1}{N}\sum_{k=1}^{N}\left(\frac{y_{n,k} - \tilde{y}_{n,k}}{r_{tol}|y_{n-1,k}| + a_{tol}}\right)^2\right)^{1/2}, \quad (12)$$

where by $y_{n,k}$ we denote the k-th entry of the discrete vector y_n, and where N corresponds to the total number of entries in y_n (the total number of dislocation degrees of freedom). We note that since this norm incorporates the relative and absolute tolerances, r_{tol} and a_{tol}, a norm value less than one indicates a "small" value with respect to the desired integration accuracy.

We use this local truncation error estimate in two ways. First, we determine whether the error in a given step is acceptable or if the step must be repeated, by requiring that each step satisfy $e_n \leq 1$. Second, we predict the maximum h that will meet the error tolerances, which we use both when taking a new step and when repeating a step having too much error. We keep track of the three most-recent error estimates, e_n, e_{n-1}, e_{n-2}, and predict the maximum prospective step size as one of

$$h' = h_n\, e_n^{-0.58/p}\, e_{n-1}^{0.21/p}\, e_{n-2}^{-0.1/p}, \quad (13)$$

$$h' = h_n\, e_n^{-0.8/p}\, e_{n-1}^{0.31/p}, \quad (14)$$

$$h' = h_n\, e_n^{-1/p}, \quad (15)$$

corresponding to the *PID*, *PI*, and *I* time-adaptivity controllers, respectively [20, 21, 22, 23], and where p is the order of accuracy for our embedded method. Upon selection of our candidate step size h', we then set the new step size h_{n+1} as

$$h_{n+1} = \min\{ch', \, \eta_{max}\, h_n\}. \tag{16}$$

Clearly, this temporal adaptivity approach results in a number of parameters that may be tuned to optimize solver performance. In this paper, we use the default parameters within the ARKode solver library (see Section 5). These values have been chosen to perform well on a wide range of test problems, matching the parameters chosen in the widely-used adaptive time integration solver, CVODE [24]. Specifically, we use the safety factor $c = 0.96$, that provides a somewhat more conservative step size (96% the size) than the value predicted by the estimates (13)-(15). In addition, the solver adjusts this maximum growth factor η_{max} based on the success or failure of the preceding step. We allow the step size to grow by at most a factor $\eta_{max} = 20$ at normal steps, although in practice the adaptive error controller limits the step size on nearly all steps. For the first step, which is typically chosen to be small to help get things started, we allow a more aggressive maximum growth factor of $\eta_{max} = 10^4$. However, when the attempted step size was too large, we reduce this growth factor to force smaller steps during difficult phases of evolution: after a step with too much error we set $\eta_{max} = 0.3$, and after a solver has failed to converge we set $\eta_{max} = 0.5$. We note that only this last value of $\eta_{max} = 0.5$ deviates from the ARKode default; here it was chosen to match the corresponding default value in `ParaDiS`, to result in more fair comparisons between solvers.

As is common with IVP solvers [24, 25], to begin the adaptive approach, we conservatively choose the first step so that an explicit Euler method would achieve the desired accuracy level. To this end, we choose $h_0 = \left(\frac{2}{\|y''\|_{WRMS}}\right)^{1/2}$, where we estimate y'' using a finite difference approximation of the right-hand side,

$$y'' \approx \frac{1}{\delta}\left[f\left(t_0+\delta, y_0+\delta f(t_0,y_0)\right) - f(t_0,y_0)\right].$$

As will be further described in the next section, an additional source of flexibility in these implicit solvers is that due to their iterative nature, the nonlinear systems $g(z) = 0$ need only be solved approximately. To this end, we define the factor ϵ_n as the nonlinear solver convergence tolerance,

$$\|g(z)\|_{WRMS} \le \epsilon_n. \tag{17}$$

Because the error tolerances are embedded in the weights applied in the norm for the error test, if each step satisfies the constraint that $e_n \le 1$, then the local error of the solutions should be within allowable error. Thus, the actual value used for ϵ_n may be modified (while staying below 1) without affecting the accuracy of the overall solve. As such, ϵ_n may be tuned to increase the computational efficiency of the method, as will be further discussed in Section 6.

In regards to implementation, our DIRK method requires slightly more storage than the trapezoid method. In general, our basic integrator and predictor algorithms require $(s+10)$ vectors, corresponding to: the stage right-hand sides $f(t_n + c_i h_n, z_i)$, an error-weight vector for norm calculations, the solutions and right-hand side vectors at the current and previous step (y_{n-1}, y_n, $f(t_{n-1}, y_{n-1})$ and $f(t_n, y_n)$), and 5 temporary vectors used in implementing the algorithms described above. By comparison, the trapezoid integrator requires two vectors. We note that these storage requirements ($s + 10$ vs 2) are in addition to any memory required for the nonlinear solvers, that apply equally to both methods, as will be discussed in the next section. Lastly, we point out that all of these vectors store only the positions of each dislocation node, which corresponds to a relatively insignificant fraction of the overall storage used within the `ParaDiS` data structures.

Within this paper, we investigate three DIRK methods. The first is Billington's singly-diagonally implicit Runge-Kutta (SDIRK) method [26], where the solution y_{n+1} has global accuracy $\mathcal{O}(h^3)$ and the embedding $\tilde{y}_{n+1}$ has global accuracy $\mathcal{O}(h^2)$. The second method we use is the globally $\mathcal{O}(h^4)$ (with $\mathcal{O}(h^3)$ embedding) SDIRK method given on page 100 of [27]. Our third DIRK algorithm is the higher-order ESDIRK 5/4a method by Kværnø [28]. In that seven-stage method (six are implicit), the solution y_{n+1} has global accuracy $\mathcal{O}(h^5)$ and the embedding $\tilde{y}_{n+1}$ has global accuracy $\mathcal{O}(h^4)$. We do not reproduce the coefficients for these methods here due to space limitations, however the coefficients defining each method are available in the cited references.

We selected the above three methods because out of all the DIRK methods we were aware of, these included embeddings and required the least number of implicit stages per step. As such, we felt that these promised to be the most efficient candidates available, due to their computationally inexpensive temporal error estimates and minimal number of implicit solves required per step.

4. Nonlinear Solvers

The implicit integrators discussed above all require one or more solves of a nonlinear, algebraic system at each time step. These systems are given as either (6) for the trapezoid integrator or as (11) for the DIRK integrators.

4.1. Fixed Point Iteration

The simplest method to solve these systems is a fixed point iteration. The iteration proceeds from an initial iterate, $y^{(0)}$, and updates with

$$y^{(k+1)} = \phi(y^{(k)}). \tag{18}$$

For both the trapezoid and DIRK integrators in the previous section, we may construct the fixed point iteration function as

$$\phi(y) \equiv y - g(y), \tag{19}$$

such that a y satisfying $g(y) = 0$ also satisfies $y = \phi(y)$, and is hence a fixed point of ϕ. As will be discussed later on, if h_n is sufficiently small this iteration is linearly convergent, meaning that the error of the $k+1$ iterate is a constant (less than 1) times the error of the previous iterate. As a result, convergence can be slow when that constant is close to 1.

In the 1960s, an acceleration method for fixed point iterations was developed in the electronic structures community [12] and only recently has been used in other application areas, where it has resulted in significant speedups of the original fixed point method [13, 14, 15, 29]. The *Anderson accelerated* fixed point method is formulated as

Algorithm AA: Anderson Acceleration

```
Given $y^{(0)}$ and $m \geq 1$.
Set $y^{(1)} = \phi(y^{(0)})$.
For $k = 1, 2, \ldots,$ until $\|y^{(k+1)} - y^{(k)}\| < \epsilon_n$
    Set $m_k = \min\{m, k\}$.
    Set $F_k = [f_{k-m_k}, \ldots, f_k]$, where $f_i = \phi(y^{(i)}) - y^{(i)}$.
    Determine $\alpha^{(k)} = \left[\alpha_0^{(k)}, \ldots, \alpha_{m_k}^{(k)}\right]^T$ that solves
        $\min_\alpha \|F_k\alpha\|_2$ such that $\sum_{i=0}^{m_k} \alpha_i = 1$.
    Set $y^{(k+1)} = \sum_{i=0}^{m_k} \alpha_i^{(k)} \phi(y^{(k-m_k+i)})$.
```

Basically, this algorithm uses a linear combination of up to m prior fixed point function values rather than just the last one to update the solution. The specific combination used is the one that would minimize the fixed point residual if ϕ were linear. This algorithm requires a least squares solution of size $N \times m_k$ at each iteration in order to solve the minimization problem. Thus, the expense of the algorithm increases as the number of iterations grows, with an added $\mathcal{O}(Nm_k)$ extra operations per iteration over the standard fixed point method. We further note that in this algorithm, the norm used to measure convergence is particular to the time integration approach, as previously discussed in Section 3.

The convergence rate of the Anderson accelerated fixed point method has not yet been fully established. However, recent work has shown that for linear ϕ and under certain conditions, the method is equivalent to the GMRES iterative method for non-symmetric linear systems [29]. In addition, for nonlinear systems the method has been shown to be equivalent under certain other conditions to a variant of the quasi-Newton method [30]. Lastly, recent work has shown that the accelerated fixed point method will converge if the fixed point method does [31].

If $\phi(y)$ from (18) satisfies the Lipschitz continuity condition,

$$\|\phi(y) - \phi(y^*)\| \le M\|y - y^*\| \tag{20}$$

for all y, y^*, and the Lipschitz constant M is independent of y and y^* and satisfies $0 \le M < 1$, then the iteration given by (18) converges to a unique solution of the fixed point problem. Note that ϕ is not Lipschitz continuous unless f is. If L is the Lipschitz constant of f, then $M = \frac{h_n}{2}L$ for the trapezoid method and $M = h_n A_{ii} L$ for the DIRK methods, so the time step sizes must meet the condition

$$h_n < \frac{2}{L} \quad \text{[trapezoid]}, \quad \text{or} \quad h_n < \frac{1}{A_{ii}L} \quad \text{[DIRK]} \tag{21}$$

in order to ensure convergence of the fixed point method [9]. In general, $L \gg 1$ for stiff systems, so when the system is stiff, time steps may have to be quite small to ensure convergence, and Newton's method is often used as an alternate.

4.2. Newton's Method

Newton's method finds the solution, y, such that $g(y) = 0$. At the k-th iteration, Newton's method forms an update to its current iterate by finding a root of the linear model

$$g(y^{(k+1)}) \approx g(y^{(k)}) + J_g(y^{(k)})(y^{(k+1)} - y^{(k)}), \tag{22}$$

where $J_g(y^{(k)})$ is the Jacobian of g evaluated at $y^{(k)}$. For large-scale problems, such as arise in DD models, this linear system is solved inexactly with an iterative method. Hence, the method is an inexact Newton method [32], and (22) is solved approximately to a specified tolerance. The following is a brief outline of the resulting inexact Newton method.

Algorithm INI: INEXACT NEWTON ITERATION

```
Given $y^{(0)}$.
For $k = 0, 1, \ldots,$ until $\|g(y^{(k)})\| < \epsilon_n$
    For $tol_L \in [0, 1)$, approximately solve $J_g(y^{(k)})\Delta y^{(k)} = -g(y^{(k)})$
      so that $\|J_g(y^{(k)})\Delta y^{(k)} + g(y^{(k)})\| \le tol_L \|g(y^{(k)})\|$.
    Set $y^{(k+1)} = y^{(k)} + \Delta y^{(k)}$.
```

As with the Anderson acceleration algorithm, the integrators use different norms to determine nonlinear and linear solver convergence in this algorithm, with the trapezoid integrator using the maximum norm and the DIRK integrator using the WRMS norm. Furthermore, the DIRK integrator sets the linear solver tolerance $\mathrm{tol}_L = \epsilon_l\, \epsilon_n$, whereas the trapezoid integrator uses $\mathrm{tol}_L = \epsilon_l$, based on the input value ϵ_l. Many strategies have been employed in order to choose ϵ_l to minimize "oversolving" the successive linear systems while maintaining adequately fast convergence of the overall nonlinear scheme; for some strategies in use, see [11, 33].

Krylov methods, such as GMRES [34], are particularly attractive for solving the linear systems in DD models since they do not require formation of the Jacobian matrix as only the action of that matrix on a vector is needed. This matrix-vector product can be approximated through a finite difference computation

$$J_g(y)v \approx \frac{g(y+\epsilon v) - g(y)}{\epsilon}. \tag{23}$$

Compared with fixed point iteration, Newton's method additionally requires computations with the Jacobian matrix, although within a Newton-Krylov framework this additional requirement translates only to extra nonlinear function evaluations as part of the finite difference calculation above. We caution that, for DD calculations, each function evaluation does include computation of all forces and thus these evaluations are costly. However, Newton's method has a faster (quadratic) rate of convergence, meaning that the error of the $k+1$ iterate is a constant times the square of the error in the previous iterate. Thus, once the iterates get close to the solution, convergence is rapid.

5. Software

The software framework used to test the integrators and solvers discussed above is comprised of the `ParaDiS` dislocation dynamics simulator and the SUNDIALS suite including KINSOL and ARKode for nonlinear solvers and multi-stage ODE integrators, respectively.

5.1. The Parallel Dislocation Simulator (ParaDiS)

`ParaDiS` [1] is a large scale dislocation dynamics simulation code to study the fundamental mechanisms of crystal plasticity. It was originally developed at Lawrence Livermore National Laboratory (LLNL). It is written in C (with a little C++) and uses the MPI library for communication between processors. It runs routinely on 100-1,000 processors and scalability on 132,000 processors of BlueGene/L has been demonstrated [2]. The code can be downloaded freely at paradis.stanford.edu. The computational approach to dislocation dynamics used in `ParaDiS` is fairly simple. In it, one introduces dislocation lines discretized in linear segments into the computational volume and lets them interact and move in response to the forces imposed by external stress and inter-dislocation interactions. The native time integrator and solver within ParaDiS is the trapezoid method with the standard fixed point nonlinear solver. In this work, we interfaced the SUNDIALS suite with `ParaDiS`, allowing for the use of Anderson accelerated fixed point or inexact Newton's method for solving the nonlinear systems within each step of the native trapezoid integrator, or the use of diagonally implicit Runge-Kutta integrators.

5.2. Suite of Nonlinear and Differential-Algebraic Equation Solvers (SUNDIALS)

The SUNDIALS suite of codes is a freely available package providing robust time integrators and nonlinear solvers designed to be easily incorporated into existing simulation codes [35]. Developed at LLNL, SUNDIALS includes the CVODE and IDA packages of multistep ODE and DAE integrators, respectively, and their forward and adjoint sensitivity-enabled versions, CVODES and IDAS, and KINSOL for solution of nonlinear algebraic equations. The newest version of SUNDIALS also contains the ARKode package with implementations of Runge-Kutta time integrators for ODEs. The packages within SUNDIALS are written in C, and provide interfaces for programs written in C++ and Fortran. SUNDIALS is written in a data-structure-neutral manner, allowing for the use of user-supplied vector kernels and linear solvers that may be optimally tuned for specific applications. In addition, SUNDIALS may utilize any of a set of generic vector kernels (serial, parallel, threaded) or linear solvers (GMRES, BiCGStab, TFQMR, PCG, FGMRES, direct, banded, LAPACK) that are supplied with the suite. Table 1 includes a quick reference for the codes and methods used in the SUNDIALS Package.

The accelerated fixed point and Newton nonlinear solvers used in the following tests are from the KINSOL package within SUNDIALS. KINSOL is an evolution of the original Newton-Krylov solver code for PDEs, NKSOL [36]. The newest version of the package will contain the Anderson accelerated fixed point method as a derivative-free nonlinear solver alternative to Newton's method.

The adaptive DIRK methods used in this work are from the ARKode solver library [37], that is also available in the newest version of the SUNDIALS suite. This library provides a flexible framework for adaptive integration of systems of initial value problems using explicit, implicit, or mixed implicit-explicit additive Runge-Kutta methods. While distributed as a component within SUNDIALS, ARKode is developed independently at Southern Methodist University.

Table 1. Definitions of code names and methods used in SUNDIALS

Name	Meanings
SUNDIALS	Suite of Nonlinear and Differential/Algebraic equation solvers
ODE	Ordinary differential equation
DAE	Differential/Algebraic equation
BDF	Backward Differential Formulae, used for multistep solution of ODE and DAE systems
RK	Runge-Kutta method, used for multistage solution of ODE systems
CVODE	ODE solver in SUNDIALS based on BDF methods
ARKode	Additive Runge-Kutta ODE solver in SUNDIALS
IDA	Implicit differential/algebraic equation solver in SUNDIALS
KINSOL	Nonlinear equation solver in SUNDIALS
CVODES, IDAS	Sensitivity-enabled ODE and DAE solvers in SUNDIALS
DIRK	Diagonally implicit Runge-Kutta methods available in ARKode

6. Numerical Results

In the following subsections we present results from three test problems, each looking at a different level of complexity. The first, the Frank-Read source problem, is a serial test giving an indication of whether the new solver or integrators will provide larger time steps. The second problem, the cold start test case, starts with a simple dislocation line configuration and tests whether the new methods will lead to faster run times. It also provides a first place to start narrowing down possible values for parameters in the new methods. The last test, the warm start test case, runs a problem that has already evolved and includes complex segment interactions, and is more indicative of solver performance on "typical" simulations.

In the following results, "Trap" and "DIRK" refer to simulations run using either the trapezoid integrator or a Diagonally Implicit Runge-Kutta integrator respectively. The number following "DIRK" gives the order of accuracy of the method for the simulation. The nonlinear solver used with the integrator is labeled "FP" for the standard fixed point solver, "AA" for the Anderson accelerated fixed point solver and "NK" for the Newton-GMRES solver. The maximum number of nonlinear iterations is indicated by the integer following the letter "I", and for the accelerated fixed point solver the number of saved residual vectors follows the letter "V". For example, "Trap AA I3 V2" indicates that the simulation was run using the trapezoid (Trap) integrator with the Anderson accelerated (AA) fixed point solver allowing up to 3 nonlinear iterations (I3) and 2 saved residual vectors (V2).

Furthermore, in many of these results we consider the "speedup" in the simulation due to the use of advanced solvers and/or higher-order time integration methods. In these results, if the native `ParaDiS` solver required wall-clock time T_{native} for a given problem, and a competing solver required wall-clock time T for the same problem, then we compute the speedup resulting from use of the new solver as

$$\%\ \text{Speedup} = 100\,\frac{T_{\text{native}} - T}{T_{\text{native}}}.$$

Additionally, the number of time steps reported for a method in the results below refers to the total number of successful steps unless otherwise noted.

For each set of runs below we state the simulation tolerance used. In the case of the trapezoidal solvers, the tolerance represents the variation in the nodal locations below which we do not care to distinguish. In particular, the maximum difference in each nodal location computed between two consecutive iterations is measured, and when that maximum is below the tolerance, then the iterative process is finished. As such, for the trapezoid solvers, the simulation tolerance refers to ϵ_n in (7) indicating how accurately we solve the trapezoid nonlinear residual equation (6), i.e. how accurately we solve the discrete problem, which itself only approximates the true continuous solution. This difference between the true differential equation and the trapezoidal approximation is never measured or controlled within the solver.

For ARKode, normally two tolerances are specified, as shown in (12). To mimic the trapezoidal solver's control over only absolute tolerance, we disabled the ARKode relative tolerance by taking its value to be 10^{-30}. We point out, however, that the term "tolerance" has different meanings for the trapezoid and ARKode solvers. In ARKode the tolerance refers to how accurately the computed solution solves the underlying continuous differential equation, i.e. how accurately we approximate the true solution to the problem. Thus for DIRK solvers, the simulation tolerances below refer to a_{tol} in (12). Subsequent tolerances for all nonlinear and linear solves within ARKode are then taken to be specified fractions of this absolute solution tolerance. As such, even with the same "tolerance," the trapezoid and ARKode solvers inherently measure different types of error. However to be as fair as possible in the tests that follow, we apply the same tolerance to both solvers.

We note that in all runs using the Newton-Krylov method below, the size of the Krylov subspace in GMRES was taken to be 5.

Lastly, in addition to demonstrating how more advanced (and potentially costly) nonlinear solvers and time integration strategies may be used on dislocation dynamics simulations, we wish to provide recommendations regarding optimal options and parameters for these solvers. Therefore, as our test problems progress from simplest to most difficult, our choices of solver options and parameters will narrow as we determine non-optimal choices for these problems. While we do not purport that our final recommended solvers and options will be optimal for all dislocation dynamics simulations, we believe that this exploration and narrowing will provide insight to practitioners wishing to invest in new algorithms.

All runs were conducted on the LLNL Cab machine. Cab is a Linux cluster with 1,296 compute nodes, each containing two Intel Xeon 8-core E5-2670 chips and 32 GB of memory. The CPUs run with a 2.6 GHz clock. The Frank-Read Source problem was run on a single core, and all other tests were done on a single node utilizing 16 cores in parallel with MPI.

6.1. Frank-Read Source

Simulations with the Frank-Read source serve as an initial starting point to gauge the viability of these mathematical techniques in dislocation dynamics simulations. The initial setup consists of a single dislocation between two pinned end points under a constant external stress or a constant external strain. The external forces cause the dislocation line to bow out. Eventually the dislocation line curls, reconnecting with itself and giving rise to two concentric dislocations. This process repeats, producing a spreading set of concentric dislocations as time goes on, as shown in Figure 1.

Results from the Frank-Read source are for a single serial run under a constant strain rate of 1 s^{-1} to a final time of 50 μs with a simulation tolerance of $1.0|b|$ (where $|b|$ is the norm of the Burgers vector). Simulations use the native ParaDiS trapezoid integrator with the standard fixed point solver, the trapezoid integrator using the Anderson accelerated fixed point or Newton-GMRES solver, and the third through

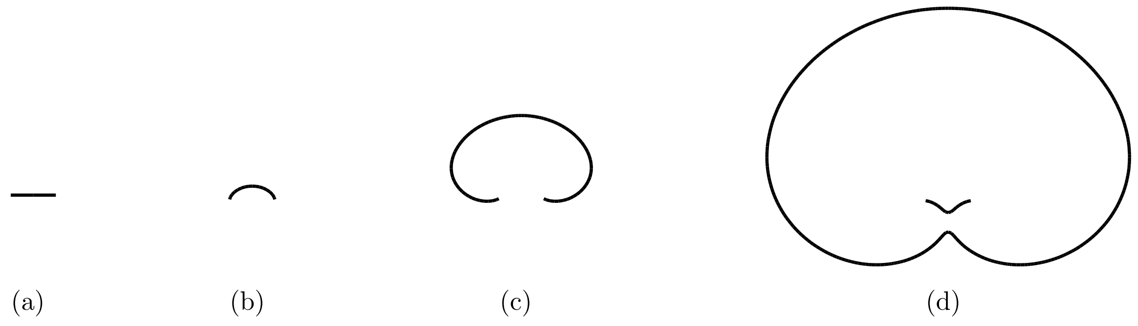


Figure 1. (a) A Frank-Read source is a straight dislocation segment anchored on two nodes. (b)-(c) As an external stress is applied to the dislocation segment, the segment starts to bow out between the two pinning points. (d) If the stress is sufficient, the dislocation expands rapidly until it creates a dislocation loop. After a loop is created, the initial segment is restored and can bow out again.

fifth-order DIRK methods described in Section 3.2. The nonlinear systems within the DIRK methods are solved using either the Newton-GMRES method or the Anderson accelerated fixed point solver. Methods using the Newton-GMRES solver utilize the finite difference Jacobian-vector product given in (23). With each integrator and solver, the number of nonlinear iterations is varied, and for the accelerated fixed point solver, different numbers of saved residual vectors are tested.

For this simple test problem, all of our advanced nonlinear solvers and higher-order time integrations perform well. Since we begin with a large array of solvers and options, we present results for three classes of methods separately. First, in Figure 2 we plot the total number of time steps required for a set of trapezoid-based solvers. Here, we compare the native solver, consisting of a fixed-point nonlinear solver with two iterations, the fixed-point solver but allowing three iterations, the Anderson-accelerated solver with two through four iterations (using an acceleration space with one fewer iteration), and the Newton-GMRES solver using from one through five nonlinear iterations and linear solver tolerance factor ϵ_l of 0.1 and 0.5.

From these results some conclusions are immediately clear. First, the performance of all three methods improves as the allowed number of iterations is increased; and use of only a single nonlinear iteration gives rather poor results. This is understandable since in these trapezoidal methods the time step size is increased/decreased purely due to convergence/divergence in the nonlinear solver, irrespective of the resulting temporal error in the computed solution. Second, as expected from their predicted convergence theory, when allowing multiple iterations the Newton-GMRES solver performs the best, followed by the accelerated fixed-point solver, and with the basic fixed-point solver coming in last. Finally, we see that the Newton-GMRES solver performs best on this problem when the inner linear solver tolerance factor is relaxed slightly to $\epsilon_l = 0.5$.

In Tables 2 and 3 we present the total required time steps for a variety of diagonally implicit Runge-Kutta solvers. Here, we vary the method order from three through five, the number of allowed nonlinear iterations from two through four, the nonlinear

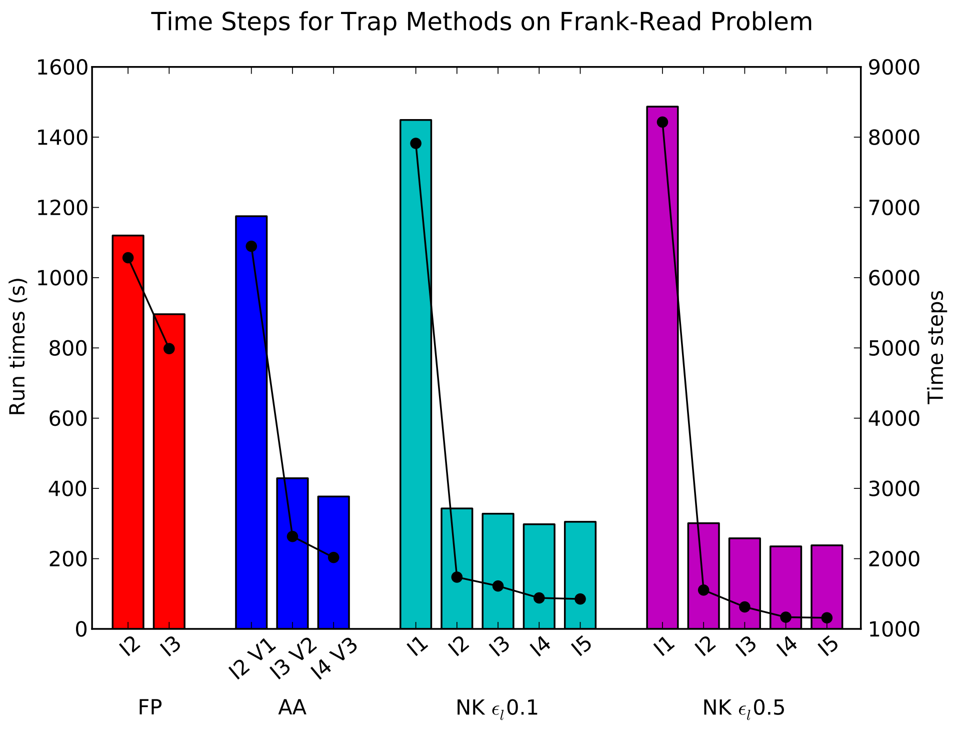


Figure 2. Run time and number of time steps for the trapezoid integrator using the FP, AA and NK solvers on the Frank-Read source problem to a final time of 50 μs. Recall ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. For each method, bars correspond to run times and the line plot corresponds to the time steps. We note that the NK solvers using 4 or 5 iterations and $\epsilon_l = 0.5$ required less than 1/5 the number of time steps of the native FP I2 solver and achieved a 79% speedup.

tolerance factor ϵ_n from 0.1 to 0.5 to 1.0, and for the Newton-GMRES nonlinear solver we also vary the GMRES tolerance convergence factor ϵ_l from 0.1 to 0.5.

Unlike the trapezoid-based solvers, the use of a fully adaptive method that includes control over temporal solution error gives rise to somewhat less obvious results. However, we can illustrate a few key trends. First, all methods generally require fewer time steps when using a larger nonlinear tolerance factor, i.e. $\epsilon_n = 0.5$ or 1.0. Second, these methods generally use fewer steps with an increased number of nonlinear iterations. Moreover, when comparing the best solvers from Table 2 with the best solvers from Table 3, it is clear that for the DIRK solvers on this problem, there is not a significant benefit to using the more powerful (and costly) Newton solver.

Perhaps more importantly for the DIRK methods is that, at least for the Frank-Read problem, some solver options have little discernible effect on the number of required time steps. First, on this problem we see little difference between the choices of linear tolerance factor ϵ_l. In addition, it is unclear whether the additional cost per step of DIRK4 or DIRK5 is worthwhile, since they do not require significantly fewer steps than DIRK3. This finding implies that the underlying dislocation dynamics system may not be sufficiently smooth as a function of time to warrant the use of methods of order

Table 2. Run time (seconds) and number of time steps for 3rd through 5th order DIRK integrators on the Frank-Read source problem using the Anderson accelerated (AA) fixed point solver to a final time of 50 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17). The native ParaDiS solver had a run time of 1120s and required 6,284 time steps for the same problem, and the DIRK solvers with $\epsilon_n = 1.0$ took as little as 1/49 as many steps with a maximum speedup of 98% (DIRK3 AA I4 V3).

	$\epsilon_n = 0.1$		$\epsilon_n = 0.5$		$\epsilon_n = 1.0$	
Method	Run time	Steps	Run time	Steps	Run time	Steps
DIRK3 AA I2 V1	172	797	98	460	76	355
DIRK3 AA I3 V2	502	3011	53	243	31	139
DIRK3 AA I4 V3	55	239	49	218	28	127
DIRK4 AA I2 V1	146	652	84	373	69	293
DIRK4 AA I3 V2	547	3360	45	186	50	198
DIRK4 AA I4 V3	55	220	49	195	50	194
DIRK5 AA I2 V1	206	842	91	387	69	288
DIRK5 AA I3 V2	105	381	64	256	180	655
DIRK5 AA I4 V3	349	1045	260	844	35	128

greater than three; we will investigate this further in the tests that follow.

As a final note on this problem, nearly all of the more advanced solvers achieved speedups over 60% on this problem, with most tested methods achieving 70-90% speedup. One even achieved a run time decrease of 98% (DIRK3 AA I4 V3).

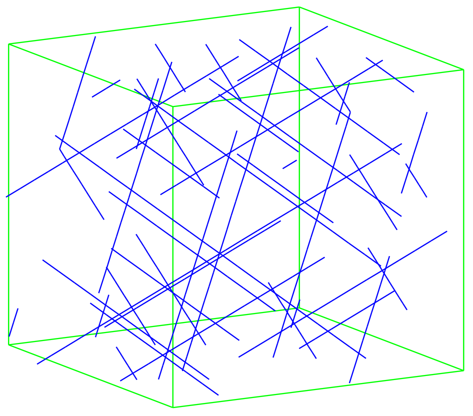
6.2. Cold Start Test Case

The Frank-Read source simulation does not do all the topological operations a larger simulation would do. Remeshing the dislocation segments is probably the only topological operation made in the Frank-Read source simulation apart from a collision when the source regenerates. A larger scale simulation, with several dislocation types and many dislocation segments usually exhibits the use of all topological operations. These operations tend to slow down the numerical integration procedure as they cut the time step to allow for dislocation segments collisions, annihilations and creation to happen.

The second set of simulations focuses on the dislocation dynamics of a crystal lattice starting from a simple line dislocation configuration. This test case provides a larger, more physically interesting problem to better judge the performance of the integrators and nonlinear solvers on target applications. The system consists of a BCC single crystal in a 4.25 μm^3 cube with periodic boundary conditions under constant compressive strain along the x-axis. The initial condition contains ~450 nodes forming straight line screw dislocations and is evolved for 3.3 μs. The initial and final dislocation networks are shown in Figure 3. All simulations are performed in parallel on 16 cores

Table 3. Run time (seconds) and number of time steps for 3rd through 5th order DIRK integrators on the Frank-Read source problem using the Newton-Krylov (NK) solver to a final time of 50 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17) and ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. The native ParaDiS solver took 1120s and required 6,284 time steps for the same problem. The DIRK solvers with $\epsilon_n = 1.0$ and 4 iterations took as little as 1/44 as many steps. Several methods achieved a speedup of 95% over the native ParaDiS solver.

	$\epsilon_n = 0.1$		$\epsilon_n = 0.5$		$\epsilon_n = 1.0$	
Method	Run time	Steps	Run time	Steps	Run time	Steps
DIRK3 NK I2 ϵ_l0.1	174	576	620	1636	178	535
DIRK3 NK I3 ϵ_l0.1	1396	2995	664	1676	64	208
DIRK3 NK I4 ϵ_l0.1	84	235	68	216	62	202
DIRK3 NK I2 ϵ_l0.5	670	1832	104	368	432	1289
DIRK3 NK I3 ϵ_l0.5	77	240	613	1788	608	1739
DIRK3 NK I4 ϵ_l0.5	78	242	60	188	89	270
DIRK4 NK I2 ϵ_l0.1	174	478	108	305	81	229
DIRK4 NK I3 ϵ_l0.1	96	237	80	195	72	176
DIRK4 NK I4 ϵ_l0.1	84	203	71	117	67	175
DIRK4 NK I2 ϵ_l0.5	144	421	106	308	436	1060
DIRK4 NK I3 ϵ_l0.5	87	231	71	187	127	227
DIRK4 NK I4 ϵ_l0.5	86	213	87	202	53	140
DIRK5 NK I2 ϵ_l0.1	202	540	136	351	113	294
DIRK5 NK I3 ϵ_l0.1	286	620	114	253	70	170
DIRK5 NK I4 ϵ_l0.1	99	215	90	213	74	161
DIRK5 NK I2 ϵ_l0.5	222	571	139	369	86	250
DIRK5 NK I3 ϵ_l0.5	1001	1868	384	575	77	185
DIRK5 NK I4 ϵ_l0.5	88	212	366	665	70	169


(a) Initial system state

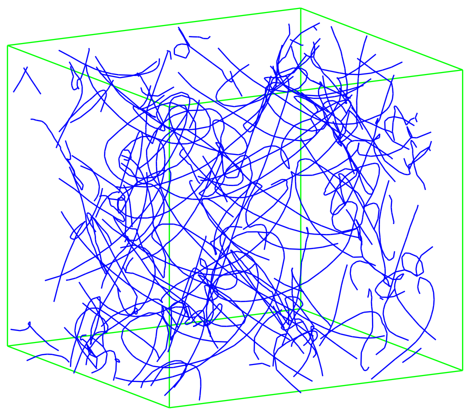

(b) System state after 3.3 μs

Figure 3. (a) The initial condition for the cold start simulations containing ∼450 nodes forming straight line dislocations. (b) The final system state after 3.3 μs with ∼2850 nodes.

with a simulation tolerance of $0.5|b|$ and a constant strain rate 10^3 s^{-1}, unless otherwise specified.

The nature of `ParaDiS` simulations is such that although two identical runs will behave the same on the macro-scale, the microstructure evolution may be different due to dynamic load balancing and negotiation of data conflicts with topological operations. There are two primary sources of these evolution differences. The first is related to the dynamic load-balancing mechanism in `ParaDiS` which is based on the wall-clock time spent in the dominant force calculations. The function used to obtain this time, however, is somewhat coarse-grained. The precise wall-clock time spent on these force calculations can therefore vary slightly between two otherwise identical runs, leading to minor changes in load-balancing. ParaDiS imposes some restrictions on the alterations that can be made to the dislocation network for segments crossing domain boundaries, so changes in the domain boundaries due to minor variations in the load-balancing lead to different micro-structure evolution. The second primary source of differences is that the order in which topological alterations are performed on the dislocation network is dependent on the nodal ID's (a pair of integers identifying both the core owning a node and the node's index within that core's node list). Since some topological operations will prevent subsequent operations on adjacent segments for the remainder of a time step, this order is significant. In particular, if a core receives incoming migrating nodes from multiple remote cores, the order in which the communications are received affects the node ID's assigned to the incoming nodes. This leads to a different sequence of alterations to the dislocation network. Thus, an alteration in the dislocation network during one run may not occur in a second identical run. Since the length of the simulation time step is sensitive to alterations in the dislocation network, the overall run-time can be affected. For larger, long-running simulations these differences average out over time, but for small initial problems run for shorter time periods, the variations can be larger.

For simulations with the cold start initial condition and native integrator, run times varied by up to 24.5% from the mean run time over a set of nine simulations. To help reduce the effects of this variability, in the ensuing results the number of time steps and wall-clock times for each solver are averaged over three identical runs.

Due to the increased nonlinearity and changing time scales of this problem, we use it to investigate potential optimizations of implicit solver parameters. As we will elaborate on in the following sections, for the trapezoidal solvers we investigate modifications to the maximum number of allowed nonlinear iterations per step and to the linear solver tolerance factor ϵ_l used with the Newton-Krylov solver. Additionally, for the DIRK integrators we use this problem to investigate optimizations of the maximum number of allowed nonlinear iterations, the nonlinear solver tolerance factor (ϵ_n in (17)), and the linear solver tolerance factor (ϵ_l in (17)) used with the Newton-Krylov solver.

As with the preceding problem, we investigate the three classes of solvers separately. First, in Figure 4, we show the run times and total time steps required for the various trapezoidal integrators. In this plot, the bars correspond with the run times for each method (left axis), and the line graph shows the number of time steps required for each method (right axis). Due to our previous results from Figure 2, in which we saw monotonically decreasing numbers of time steps as we allowed more solver iterations, we have modified the number of allowed iterations for each solver slightly. For the standard fixed point solver (FP) and the Newton-Krylov (NK) solvers, we now allow two through four nonlinear iterations, while for the accelerated fixed point solver (AA) we now allow two through six iterations, with the acceleration subspace always one smaller.

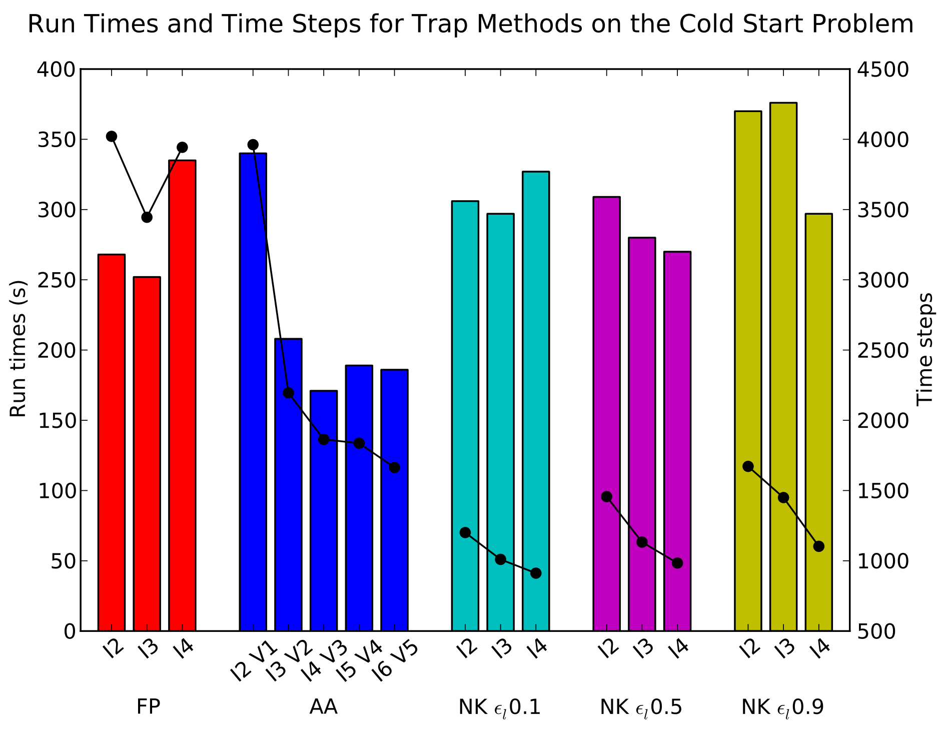


Figure 4. Run time and number of time steps for the trapezoid integrators using FP, AA and NK solvers on the cold start problem to a time of 3.3 μs. Recall ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. For each method, bars correspond to run times and the line plot corresponds to the time steps. We note that the AA solvers using at least 3 iterations achieve up to 36% speedup over the native FP I2 solver, and the NK solvers with $\epsilon_l = 0.1$ require less than 1/4 the number of steps.

Here, the results become somewhat more complex. First we consider the number of time steps (the line plot). As before the FP solver requires the most time steps, followed by the AA solver, and the NK solvers take the fewest steps. Also as before for the AA and NK solvers, the number of time steps decreases as the iterations are increased. However a few new features arise. The FP solver initially quickens with increasing iterations, but slows dramatically if too many iterations are allowed. Similarly, the NK solver requires slightly fewer time steps when the ϵ_l is smaller, corresponding with the Newton convergence theory.

However, by investigating the run times we now get a more complete picture.

Although the NK solvers require the least time steps, they require additional force evaluations to compute the finite difference Jacobian-vector products. As a result, their significantly higher cost of each iteration outweighs the benefit in reduced time steps, resulting in overall slower methods than the native FP solver. The AA solver, though, provides a happy medium of faster convergence with only a marginal increase in cost, allowing for run time decreases in comparison with FP, about 30%. Finally, we see a turning point in the AA solver run times, wherein the cost of increased allowed iterations (and acceleration subspace size) begins to outweigh the time step decrease, with the fastest overall trapezoidal solver being the AA I4 V3 method.

Moving to the DIRK integrators, in Table 4 we present results for the third through fifth order methods using the AA nonlinear solver showing both the run times and the number of integration steps. Again, all DIRK methods require significantly fewer time steps than the native solver (FP I2). However the DIRK methods require more work per step than the trapezoidal methods, since they require a nonlinear solver per stage within the step, while the trapezoidal solver requires only one such solve. As a result, the increased work per step outweighs the time step decrease for many of the DIRK solver options. As with the results in Tables 2 and 3, we see that orders four and five do not admit significantly fewer steps than order three, with the fourth order method showing slowdown for all solver options; we will therefore discontinue investigation of DIRK4 in the remainder of these results. However, investigating those options that do work well, we clearly see that the DIRK solvers perform best with nonlinear tolerance factor ϵ_n of $0.5|b|$ or $1.0|b|$. Consequently, we will discontinue investigation of $\epsilon_n = 0.1$ in all following results.

Similarly, in Table 5 we investigate the DIRK methods of orders three and five with the NK solver on this problem. As with the preceding results, all solvers required significantly fewer time steps than the native solver, but the increased work per step far outweighed this benefit, since the increased number of nonlinear solves per step is compounded by the increase in force evaluations used in the finite difference Jacobian-vector products. Accordingly, all NK-based methods slowed down in comparison with the native solver. However, investigating the best solver options for this method, we see that the optimal solver combinations consist of the third order method, using three or four nonlinear iterations, and with linear tolerance factor ϵ_l of either 0.5 or 0.9.

In summary, for the cold start problem, we find that the fastest solver is the Trapezoid integrator using the accelerated fixed point solver, with 4 allowed solver iterations and 3 saved vectors. Moreover, we find that although no Newton-based solver achieves a speedup over the native solver, all require significantly fewer time steps, with the fastest Trapezoidal solver (Trap NK ϵ_l 0.1) taking 912 steps, and the fastest DIRK solver (DIRK5 NK I4 ϵ_l 0.9) taking 386 steps.

In Figure 5, we compare the dislocation density over time in the cold start problem using the native `ParaDiS` trapezoid integrator with the standard fixed point solver, the fastest trapezoid integrator with the accelerated fixed point solver and the fastest third and fifth order DIRK methods with the accelerated fixed point solver. The different

Table 4. Run time (seconds) and number of time steps for 3rd through 5th order DIRK methods using the AA solver on the cold start problem to a time of 3.3 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17). The native ParaDiS solver required 268 seconds and 4,021 time steps for the same problem. We note that although all DIRK AA solvers require less than 1/5 the number of time steps as the native solver, only the DIRK 3 AA methods with ϵ_n of 0.5 and 1.0 are faster with speedups between 17% and 29%.

	$\epsilon_n = 0.1$		$\epsilon_n = 0.5$		$\epsilon_n = 1.0$	
Method	Run time	Steps	Run time	Steps	Run time	Steps
DIRK3 AA I4 V3	428	696	212	419	197	443
DIRK3 AA I5 V4	327	538	220	399	190	434
DIRK3 AA I6 V5	297	476	220	408	201	420
DIRK3 AA I7 V6	265	408	215	393	222	429
DIRK4 AA I4 V3	445	576	392	544	398	525
DIRK4 AA I5 V4	422	476	357	515	328	497
DIRK4 AA I6 V5	410	465	406	533	372	520
DIRK4 AA I7 V6	400	447	418	532	360	530
DIRK5 AA I4 V3	581	589	273	345	274	337
DIRK5 AA I5 V4	531	486	307	342	284	331
DIRK5 AA I6 V5	475	419	316	339	301	339
DIRK5 AA I7 V6	446	376	299	305	279	308

methods all show good agreement in the dislocation density curves throughout the simulation.

6.3. Warm Start Test Case

Our remaining computational results focus on our third, and most challenging, test problem. As with the cold start problem, this case consists of a BCC single crystal in a 4.25 μm^3 cube with periodic boundary conditions, but where the simulation begins at the end of the cold start test case with a more developed structure. This simulation starts with ∼2850 interacting dislocations and is run for either 1.1 μs or 2.95 μs from the final time of the cold start case, 3.3 μs. Thus the final simulation times are either 4.4 μs or 6.25 μs, as detailed in each set of results. The final dislocation networks for each final time are shown in Figure 6. This restarted initial condition provides insight as to how well the integrators and nonlinear solvers function on a system after dislocations have begun to interact. Tests are also conducted at different constant compressive strain rates from 10^{-1} s^{-1} to 10^3 s^{-1} along the x-axis. Again, all simulations are performed in parallel on 16 cores with a simulation tolerance of $0.5|b|$.

Similar to the cold start case, for this restarted initial condition and native integrator, the wall-clock time differs from the mean by up to 10%, again over nine runs. Thus, in the ensuing results the number of time steps and wall-clock times for

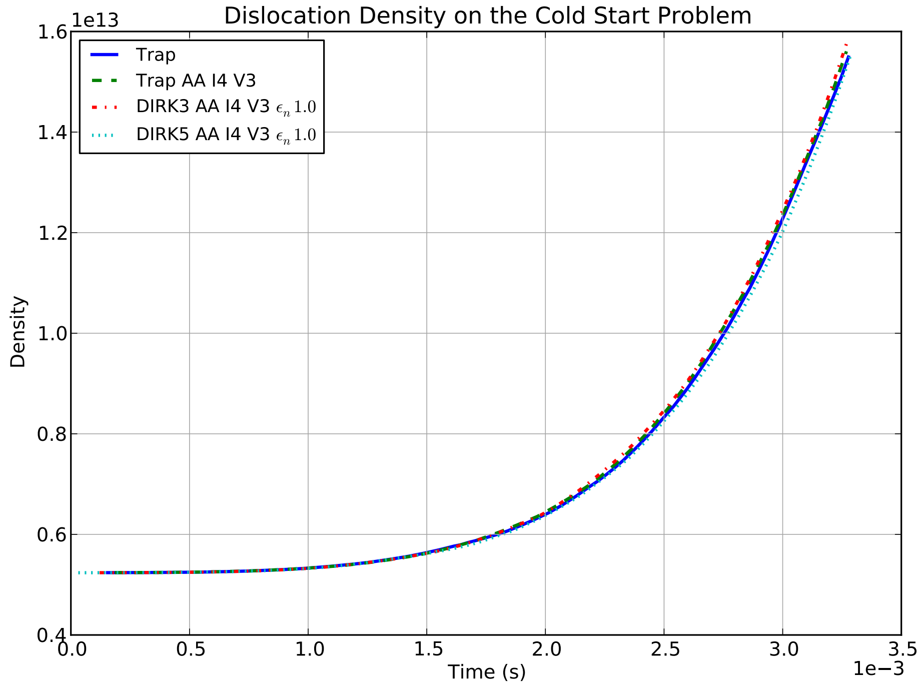
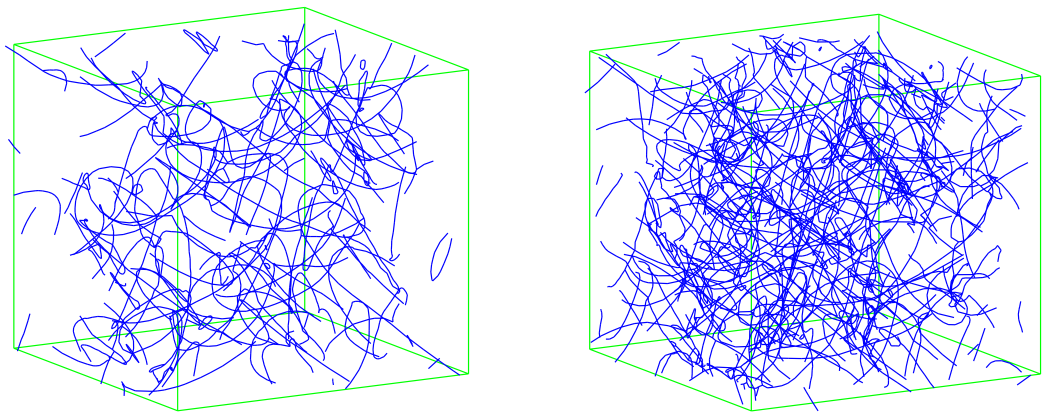


Figure 5. Dislocation density for the cold start problem using the trapezoid method with two nonlinear iterations, trapezoid using Anderson acceleration with four iterations and three residual vectors, and the 3rd and 5th order DIRK integrators with Anderson acceleration with four nonlinear iterations and three residual vectors with nonlinear tolerance factor ϵ_n 1.0. The different methods show good agreement in the density curves throughout the duration of the simulation.



(a) System state after 4.4 μs (b) System state after 6.25 μs

Figure 6. (a) The final dislocation network for the warm start test after 1.1 μs for a final simulation time of 4.4 μs containing ∼2920 nodes. (b) The final warm start system state after 2.95 μs for a final time of 6.25 μs with ∼4950 nodes.

Table 5. Run time (seconds) and number of time steps for 3rd and 5th order DIRK methods using the NK solver on the cold start problem to a time of 3.3 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17) and ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. The native ParaDiS solver required 268 seconds and 4,021 time steps for the same problem. We note that although all DIRK NK solvers require less than 1/6 the number of time steps as the native solver, none are faster due to the increased solver cost per step.

	$\epsilon_n = 0.5$		$\epsilon_n = 1.0$	
Method	Run time	Steps	Run time	Steps
DIRK3 NK I2 $\epsilon_l 0.1$	856	585	780	542
DIRK3 NK I3 $\epsilon_l 0.1$	743	485	658	462
DIRK3 NK I4 $\epsilon_l 0.1$	824	494	653	466
DIRK3 NK I2 $\epsilon_l 0.5$	802	586	588	514
DIRK3 NK I3 $\epsilon_l 0.5$	682	486	536	430
DIRK3 NK I4 $\epsilon_l 0.5$	575	423	543	428
DIRK3 NK I2 $\epsilon_l 0.9$	714	582	585	494
DIRK3 NK I3 $\epsilon_l 0.9$	596	446	540	447
DIRK3 NK I4 $\epsilon_l 0.9$	671	456	484	400
DIRK5 NK I2 $\epsilon_l 0.1$	1493	599	1195	492
DIRK5 NK I3 $\epsilon_l 0.1$	1148	475	1122	459
DIRK5 NK I4 $\epsilon_l 0.1$	1162	451	1081	432
DIRK5 NK I2 $\epsilon_l 0.5$	1284	573	953	459
DIRK5 NK I3 $\epsilon_l 0.5$	997	433	930	414
DIRK5 NK I4 $\epsilon_l 0.5$	1085	416	858	376
DIRK5 NK I2 $\epsilon_l 0.9$	1103	525	845	433
DIRK5 NK I3 $\epsilon_l 0.9$	971	420	807	384
DIRK5 NK I4 $\epsilon_l 0.9$	892	386	802	361

each solver are averaged over three identical runs. As with the preceding problem, we investigate the three classes of solvers separately.

In Figure 7, we plot both the run times and total time steps required for the various trapezoidal integrators. In this plot as with Figure 4, the bars correspond with the run times and the line graph shows the number of time steps required for each method. As before, we plot results for the FP, AA and NK nonlinear solvers, but we have slightly increased the number of allowed nonlinear iterations for the AA and NK solvers due to the increased problem difficulty. Again, the NK solver requires the fewest time steps, followed by the AA and FP solvers. Also, again the NK solver with linear tolerance factor $\epsilon_l = 0.1$ and largest allowed iterations results in the fewest time steps. Similarly, the required number of time steps for the AA solver decreases monotonically with increased iterations.

However, the differences between Figures 4 and 7 are perhaps the most interesting. First, we see that the basic FP solver performance degrades with increasing iterations,

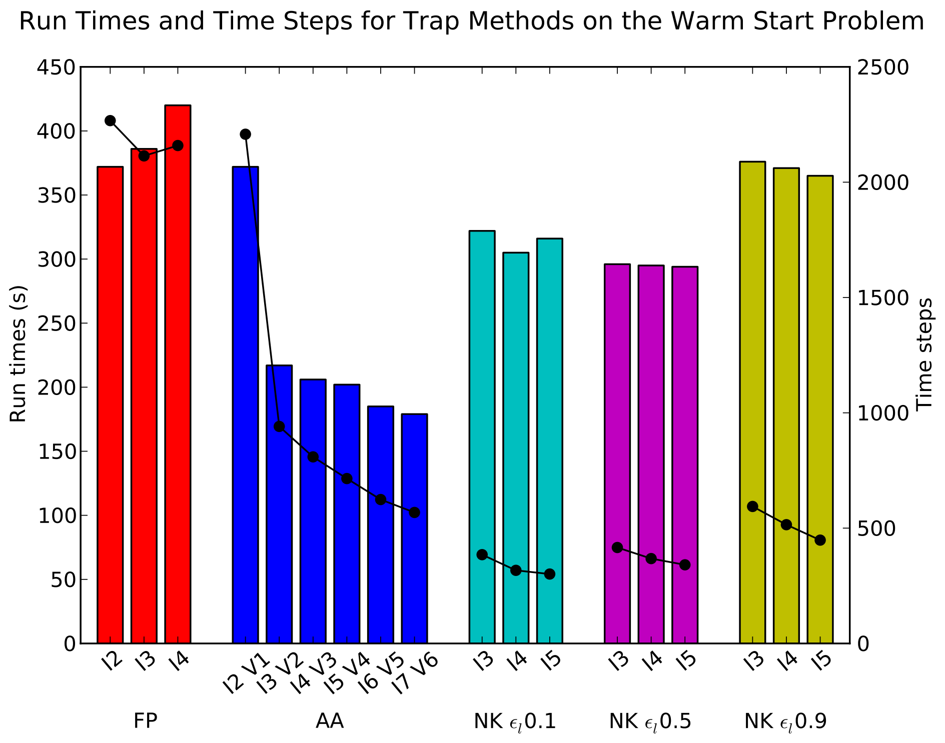


Figure 7. Run time and number of time steps for the trapezoid integrator with FP, AA and NK solvers on the warm start problem. Results used a constant strain rate 10^3 s^{-1} to a final time of 4.4 μs. Recall ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. We note that nearly all AA and NK solvers are now faster than the native solver, with the AA I7 V6 solver taking less than half the run time. As before, the NK solvers take the fewest steps, with NK ϵ_l 0.1 requiring less than 1/7 the number of time steps of the native solver.

whereas last time it improved before getting worse. Second, in nearly every case the AA and NK solvers resulted in a run time speedup over the native FP I2 solver, with the AA I7 V6 solver achieving a speedup of 52%. These results indicate that the warm start problem indeed focused on a more challenging application than the cold start problem – as dislocations grow and interact, advanced solvers are needed to handle the increasingly challenging nonlinear equations. Moreover, we see that although the NK solvers are still currently slower than the AA solvers, their dramatic reduction in time steps implies that further enhancements to how the Newton systems are created and solved may reap significant rewards, as we will elaborate on at the end of this section.

Similarly, in Figures 8 and 9 we present the same bar charts for the DIRK solvers. Specifically, in Figure 8 we show results using the AA solver for the third and fifth order DIRK methods, using nonlinear tolerance factors ϵ_n of 0.5 and 1.0, allowing from four through seven nonlinear iterations. We may immediately draw some interesting conclusions. First, all of the presented results achieved a speedup over the native FP I2 solver, with speedups ranging from 8% (DIRK 5, ϵ_n 0.5, I5 V4 and I7 V6) up to 56% (DIRK 3, ϵ_n 1.0, I4 V3). Moreover, the third order method is uniformly faster than the fifth order method, with the fifth order method not resulting in a dramatic reduction in time steps. We believe that this may be attributed to either a lack of sufficient temporal differentiability in dislocation dynamics simulations, or possibly the loose overall requested solver tolerance ($0.5|b|$) that lies outside the asymptotic convergence regimes of these methods. Moreover, although less significant, it appears that the nonlinear tolerance factor ϵ_n of 1 slightly outperforms 0.5. Finally, it again appears as though the simulation times do not depend heavily on the maximum allowed iterations, as long as these are sufficiently large to allow solver convergence.

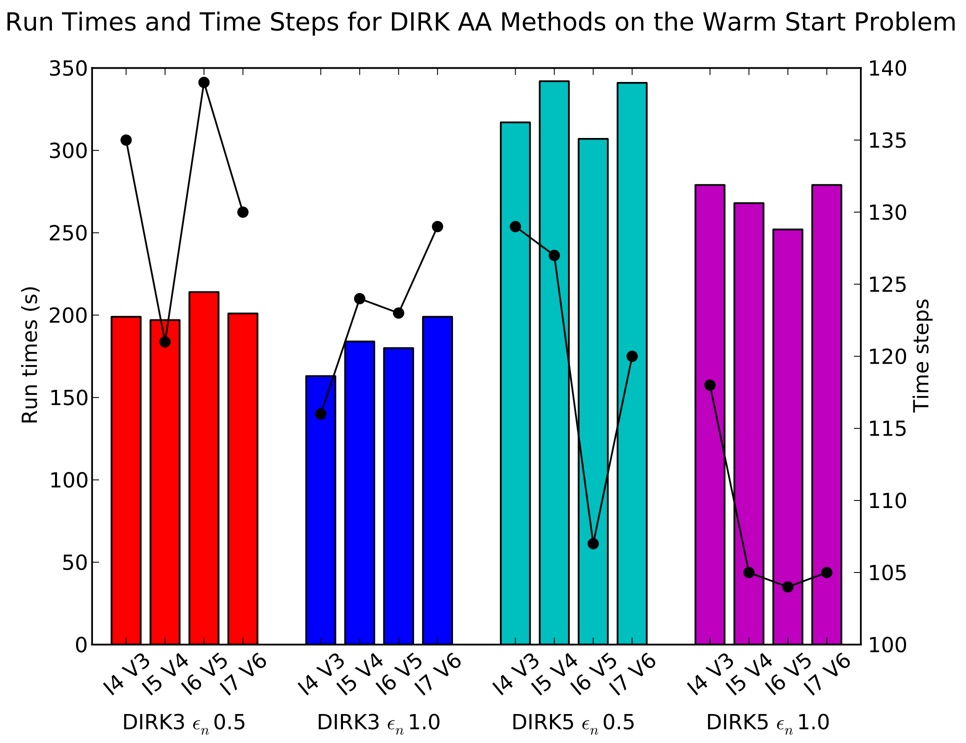


Figure 8. Run time and number of time steps for 3rd and 5th order DIRK integrators with the AA solver on the warm start problem. Results used a constant strain rate 10^3 s^{-1} to a final time of 4.4 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17). The native FP I2 solver required 372 seconds and 2267 time steps. We note that all solvers take less run time and fewer steps than the native solver, with the DIRK3 ϵ_n 1.0 I4 V3 taking less than half the run time, and DIRK5 ϵ_n 1.0 with 6 iterations taking 1/21 the time steps of the native solver.

Figure 9 presents the same data for the NK solver using the third order DIRK method, where we vary the number of allowed iterations (three or four), the nonlinear tolerance convergence factor ϵ_n (0.5 or 1.0), and the linear tolerance convergence factor ϵ_l (0.1, 0.5 and 0.9). Again, a few conclusions may be immediately drawn. First, four iterations always outperforms three in both time steps and simulation time. Second, ϵ_n of 1.0 is uniformly better in both metrics than the corresponding runs with a value of 0.5. Third, larger values of ϵ_l similarly performed better on average in both run time and time steps. Unfortunately, however, none of these results were faster than the native FP

I2 solver (372 seconds), again due to the multiple implicit stages per step and increased cost of each Newton solve.

In summary, for the above results from the warm start problem, we find that the fastest solver is the DIRK3 integrator using the accelerated fixed point solver, with 4 iterations and 3 saved vectors which lead to a 56% speedup taking 116 steps. The best Trapezoid integrator achieved a 52% speedup taking 568 steps using the accelerated fixed point solver with 7 iterations and 6 saved vectors.

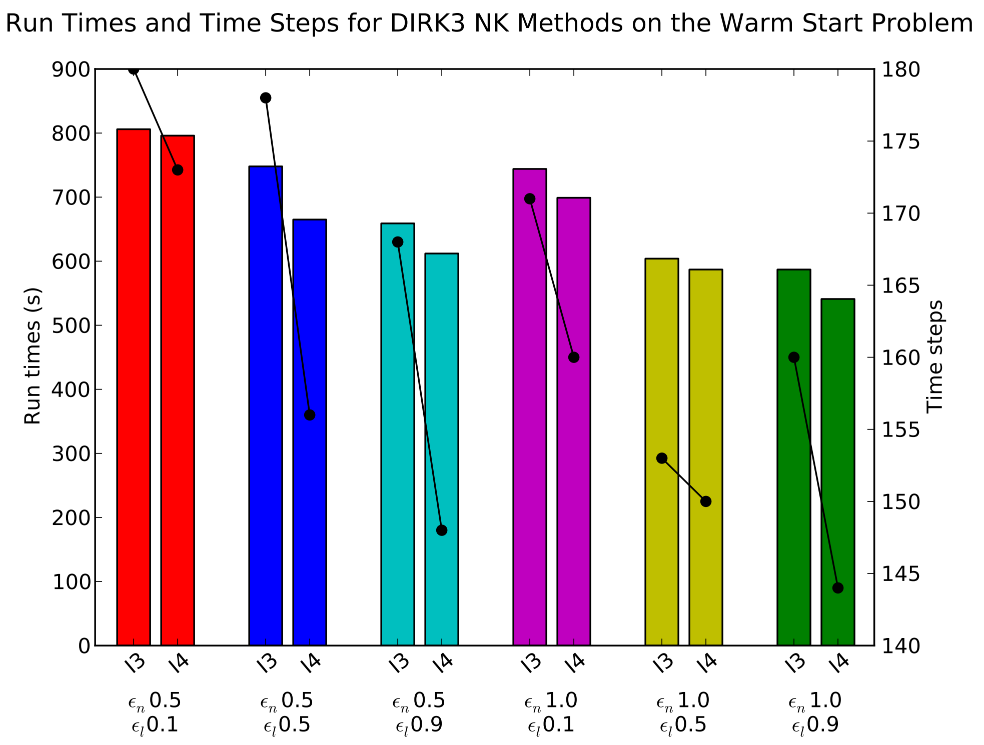


Figure 9. Run time and number of time steps for 3rd order DIRK integrator with the NK solver on the warm start problem. Results used a constant strain rate 10^3 s^{-1} to a final time of 4.4 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17) and ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. The native FP I2 solver required 372 seconds and 2,267 time steps, so the fastest of these solvers (ϵ_n 1.0, ϵ_l 0.9, I4) took 68% longer to run, but required only 1/15 the number of time steps.

As this problem is indicative of a production environment, we additionally investigated two DIRK solver options that often result in significant performance improvements: the choice of implicit predictor and the choice of time step adaptivity algorithm. As described in Section 3.2, we allow a variety of algorithms to predict the solution to each upcoming implicit solve. Thus, we tried the three predictors (a)-(c) from Section 3.2, with the third and fifth order integrators, the AA and NK solvers, and allowing either four or five nonlinear iterations. All solvers used the optimal nonlinear tolerance factor ϵ_n of 1.0, and the NK solver used the optimal linear tolerance factor ϵ_l of 0.9. Unfortunately, however, we saw no appreciable difference in either the number of time steps required (each was within 12% of the optimum) or the speedups (each was within 10% of the optimum), and indeed the optimum choice varied between each solver.

The choice of time step adaptivity algorithm, however, was somewhat more illustrative. As seen in Figure 10, we used the third order DIRK method with AA solver and either four or five nonlinear iterations, and tested three different choices of time step adaptivity algorithm. Here, PID refers to the controller in (13), PI refers to the controller in (14), and I refers to the controller in (15). These results indicate that for this problem, the simplest controller, I, gives the least overall time steps for the problem (including both accepted and discarded steps). The key differences between these algorithms is the amount of time step history that they take into consideration when determining a prospective time step size, with the I controller examining only the most-recent step, the PI controller examining two steps, and the PID examining three. These results are unsurprising, since dislocations may be added or removed between every step, which can lead to more rapid variations in the dynamical time scale as steps proceed. As a result, while a longer step history may be beneficial for many problems, it proves a slight hindrance here.

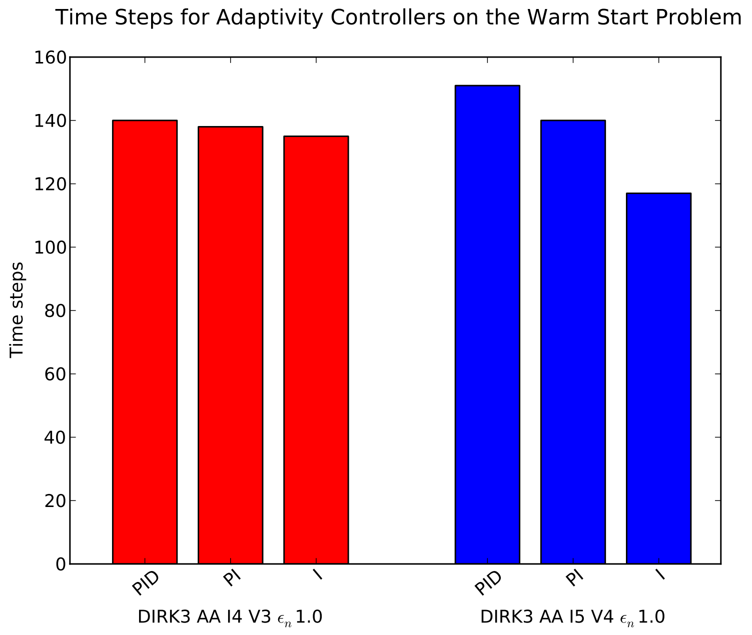


Figure 10. Number of time steps for 3rd order DIRK integrator with the AA solver on the warm start problem, using various time step controller methods. Here the PID, PI and I controllers are shown in (13), (14) and (15), respectively. Results used a constant strain rate 10^3 s^{-1} to a final time of 4.4 μs. We note that the I controller resulted in the least steps, requiring 1/19 as many as the native solver.

Next, we investigate the performance of these solvers on the warm start problem, but vary the strain rate to be 10^3, 10^2, 10^1, 10^0, or 10^{-1} s^{-1}. In order to generate results that more accurately capture the performance of each method, these simulations were run to a later final time (6.25 μs) than the preceding tests (4.4 μs). For this suite of tests, we narrow our solver choices to only the best set from each category:

- TRAP FP, the native `ParaDiS` solver, with two iterations,
- TRAP AA, the Anderson accelerated fixed point solver, using six or seven iterations,
- TRAP NK, the Newton-GMRES solver, using $\epsilon_l = 0.5$ and either three or four iterations,
- DIRK3 AA, the third-order DIRK method with the accelerated fixed point solver, using $\epsilon_n = 1.0$ and either four or five nonlinear iterations and the I controller, and
- DIRK3 NK, the third-order DIRK method with the Newton-GMRES solver, using $\epsilon_n = 1.0$, $\epsilon_l = 0.9$, four nonlinear iterations and the I controller.

Results showing the number of time steps for our final suite of solvers at these strain rates are given in Table 6. We immediately see that for every strain rate, the adaptive DIRK solvers require far fewer time steps than any other method. These are followed by the NK and then AA trapezoidal solvers, which also require a fraction of the number of time steps as compared with TRAP FP I2.

Table 6. Number of time steps for each solver on the warm start problem for various constant strain rates to a final time of 6.25 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17) and ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. We note that for all strain rates, the new solvers required fewer steps than the native FP I2 solver, with DIRK3 AA taking the fewest overall steps.

	Strain Rate (s^{-1})				
Method	10^3	10^2	10^1	10^0	10^{-1}
TRAP FP I2	10065	9137	5809	5107	5104
TRAP AA I6 V5	3225	2032	1628	1552	1348
TRAP AA I7 V6	2866	1891	1101	1260	1397
TRAP NK I3 ϵ_l0.5	1544	1004	1098	1041	1143
TRAP NK I4 ϵ_l0.5	1396	837	960	891	857
DIRK3 AA I4 V3 ϵ_n1.0	451	323	327	338	320
DIRK3 AA I5 V4 ϵ_n1.0	452	297	314	328	323
DIRK3 NK I4 ϵ_n1.0 ϵ_l0.9	483	351	361	358	359

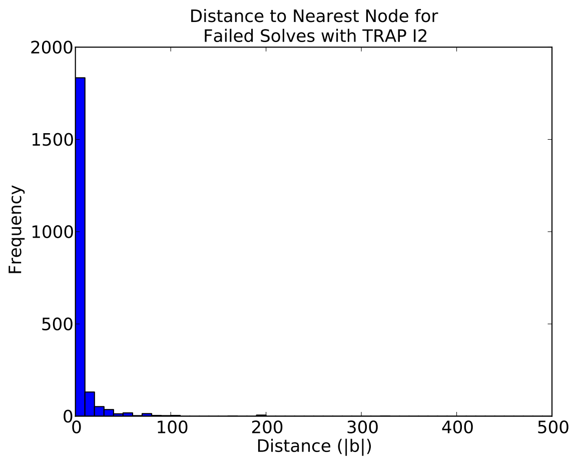
The resulting run times for these tests are given in Table 7. Here, with the exception of the DIRK method with Newton-GMRES solver, we see that all of the new solvers achieve a speedup over the native solver. Moreover, even the DIRK method with the Newton solver does not show a tremendous slowdown, indicating that Newton solver improvements for this problem may achieve strong results. Furthermore, we see that for the two fastest strain rates the adaptive DIRK method with AA solver achieves the fastest result, while for the three slowest strain rates the trapezoidal method with AA solver is fastest. Finally, we notice an interesting trend that the higher the strain rate, the better the DIRK method performs in comparison with the native solver.

Table 7. Run times (seconds) for each solver on the warm start problem for various constant strain rates to a final time of 6.25 μs. Recall ϵ_n is the nonlinear solver convergence tolerance from (17) and ϵ_l is the linear solver tolerance factor in the inexact Newton iteration. We note that for all strain rates, all new solvers except DIRK 3 NK ran faster than the native FP I2 solver, with DIRK3 AA I4 achieving the largest overall speedup. For higher strain rates, the third order integrator is fastest, but for lower strain rates, the trapezoidal method with the accelerated fixed point solver is fastest.

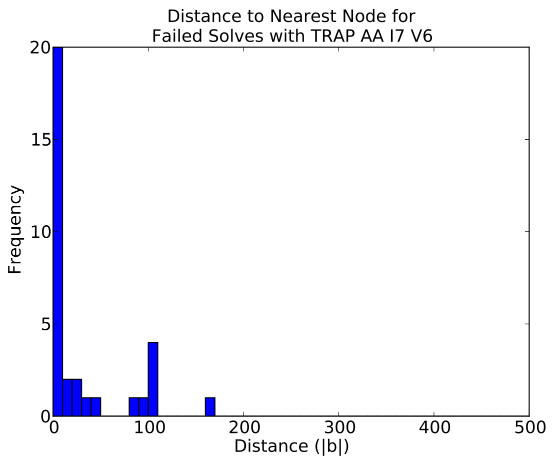
	Strain Rate (s^{-1})				
Method	10^3	10^2	10^1	1	10^{-1}
TRAP FP I2	2526	973	629	566	564
TRAP AA I6 V5	1762	437	335	316	275
TRAP AA I7 V6	1627	452	236	275	294
TRAP NK I3 ϵ_l0.5	2116	516	499	497	550
TRAP NK I4 ϵ_l0.5	2070	501	501	483	483
DIRK3 AA I4 V3 ϵ_n1.0	1011	340	330	335	320
DIRK3 AA I5 V4 ϵ_n1.0	1098	322	335	333	325
DIRK3 NK I4 ϵ_n1.0 ϵ_l0.9	2953	956	959	956	932

Finally, we analyze the structure obtained during the above warm start simulations to investigate which nodes are responsible for nonlinear solver failures. Results are taken from a single run of the warm start case with the later final time of 6.25 μs and the largest strain rate of 10^3 s^{-1}. Simulations use the trapezoid method with either the native fixed point solver with 2 nonlinear iterations or the Anderson accelerated solver with 7 nonlinear iterations and 6 residual vectors.

There exist several mechanisms that may reduce the time-step size of time integrators. In [3], short dislocation segments and segments at close proximity have been identified as the principal cause of time integrator time-step failure. For each failed nonlinear solve attempt in the trapezoid method, Figure 11 shows the distance from a failed node to nearest node in the domain. A large majority of failed nodes are within $10|b|$ of another node. Figure 12 shows the histogram of segment lengths to a failed node for all unsuccessful solve attempts. It shows that not many short dislocation segments are present in the simulations as they are fairly well distributed between 0 and $500|b|$, the maximum allowed segment size. In these simulations, the main cause of solver failure is due to nodes at close proximity to other nodes that are not necessarily connected to each other. The force per unit length between segments is inversely proportional to the distance between them. As dislocation segments get closer to each other, their interaction force becomes more non-linear, and the resulting nonlinear problems get harder to solve. This may explain why segments at close proximity failed predominantly.

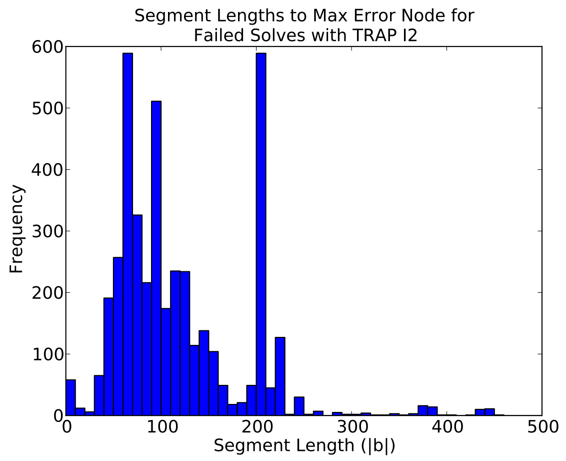


(a) TRAP I2

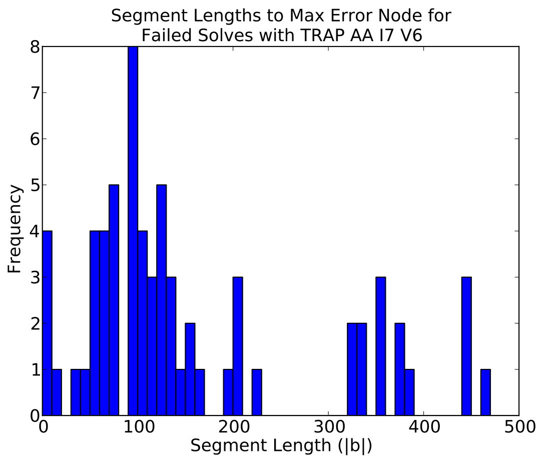


(b) TRAP AA I7 V6

Figure 11. Distance between a failed node and the nearest node in the domain for all failed solve attempts with (a) the native fixed point method and (b) the Anderson accelerated solver. A majority of failed solves are due to nodes at close proximity to one another.



(a) TRAP I2



(b) TRAP AA I7 V6

Figure 12. Distribution of segment lengths connected to a failed node for all unsuccessful solve attempts with (a) the native fixed point solver and (b) the Anderson accelerated solver. The length of the segments does not seem to be a cause of solver failure.

A Note on Newton's Method Performance. As discussed in Section 4.2, within each linear iteration the Newton-Krylov method requires either a Jacobian-vector product or an approximation to it. For these tests, we did not implement a full Jacobian as the implementation is time-consuming to complete and verify. As such, we used a finite-difference approximation to the Jacobian-vector products. This approximation results in an extra force evaluation *for each linear iteration.* These extra force evaluations are costly in terms of run time and have the potential to greatly skew the cost of Newton's method. Based on the above results, we observed that Newton's method gives a more robust solve than the original fixed point method. Since its rate of convergence is faster than fixed point, we expect a potential gain from Newton's method possibly over the accelerated fixed point if we can remove the extra force evaluations and employ an analytic Jacobian calculation.

Table 8 shows the numbers of time steps, iterations, Jacobian evaluations, and nonlinear function evaluations for the best fixed point and Newton solvers on the warm start problem used for the strain rate study. If we assume the force evaluations, thus the function evaluations, strongly dominate the run time cost, then a coarse estimate of the cost for a function evaluation is achieved by dividing the total run time by the number of function evaluations. If we then subtract the number of linear iterations from the number of function evaluations for the Newton runs and multiply the cost per function evaluation estimate by the new number of function evaluations, we get an estimated cost of a run with an analytical Jacobian. These numbers are in the estimated run time column. While these numbers do not fully account for the cost of computing the Jacobian, we note that this calculation can be added to the function evaluation in a way that makes the cost of the Jacobian a minimal added expense over a single function evaluation. These numbers show a very significant potential benefit in the use of Newton's method. Future work will include implementation of an analytic Jacobian in `ParaDiS`.

Table 8. The number of successful time steps, nonlinear iterations, linear iterations, Jacobian-vector products, function evaluations, and run times for trapezoid and DIRK integrators using the Newton-GMRES solver. The final column is the estimated run time using an analytic Jacobian in ParaDiS. Results are from the warm start problem run to a final time of 6.25 μs with a constant strain rate of 10^3 s^{-1} and a simulation tolerance of $0.5|b|$. Recall ϵ_n is the nonlinear solver convergence tolerance from (17) and ϵ_l is the linear solver tolerance factor in the inexact Newton iteration.

Method	Steps	Nonlin Iter	Linear Iter	Jv Eval	Fcn Eval	Run Time (s)	Est. Run Time (s)
TRAP FP I2	10,434	15,627	—	—	15,627	2,615	—
TRAP AA I7 V6	2,866	10,466	—	—	10,466	1,627	—
TRAP NK I3 $\epsilon_l 0.5$	1,544	3,386	5,875	7,643	12,804	2,135	1,157
TRAP NK I4 $\epsilon_l 0.5$	1,396	3,279	5,763	7,627	12,440	2,089	1,121
DIRK3 NK I4 $\epsilon_n 1.0$ $\epsilon_l 0.9$	483	4,041	10,029	13,909	20,647	2,953	1,519

7. Conclusions

Dislocation dynamics simulations present a unique set of challenges for implicit solvers and time integration methods: rapidly-changing nodal topology, non-deterministic results, and costly nonlinear function evaluations. As such, many "standard" choices for optimal integrators and solvers do not fit well in this context, including multistep instead of multi-stage methods, higher order solvers, higher order time adaptivity controllers, and higher order implicit predictors.

This paper shows performance results, in both number of time steps and run time, for an accelerated fixed point solver and higher order multi-stage time integrators on dislocation dynamics simulations. Both the solvers and integrators were employed through the SUNDIALS suite of codes [35] including KINSOL and ARKode [37]. These solvers and integrators are compared against a standard trapezoid integrator with a fixed point nonlinear solver and various parameters were investigated to determine recommended values for this application.

Results show that the Anderson acceleration method applied to the fixed point solver can lead to significant benefits in both run time and time step size. This method does not add much complexity to the solver implementation nor to run time per iteration but did accelerate the fixed point solver. Further, the diagonally implicit Runge-Kutta (DIRK) methods also led to larger time steps and significant speedups in run time. The third order DIRK methods resulted in much better performance than either the fourth or fifth order methods.

In general, for easier problems, like those earlier in time or with lower strain rates, we observed that a high order integrator was not needed and a lower order integrator with the accelerated fixed point method was enough to give very good performance in both run time and time step size. However, as problems got more difficult, the third order integrator showed very significant benefits over the lower order approaches. Furthermore, the DIRK methods employ an error estimate in their choice of time step which helps to ensure accuracy of the resulting numerical solutions. As a result, we recommend use of the DIRK methods with the accelerated fixed point solver in general.

Results further indicate that solver failures occur predominantly at nodes in close proximity to other nodes. This finding is similar to a finding in [3]. One cause of time step failure with the Trapezoid integrator was found to be failed nonlinear solves. Since the force calculations for these nodes become more nonlinear, it is not surprising that a more robust nonlinear solver was found to yield fewer failed time steps.

A basic Newton-Krylov method was tested along with the other nonlinear solvers. The implementation of this method employed a finite-difference approximation for the matrix-vector multiplies needed in the linear solver iterations. While it was simple to construct this implementation, it was not efficient and required far more function evaluations (force calculations) than would be needed with an analytical Jacobian implementation. However, results with this implementation showed that the Newton method also results in larger time steps. Future work will look at efficiency of a Newton method implemented with an analytic Jacobian.

References

[1] A. Arsenlis, W. Cai, M. Tang, M. Rhee, T. Oppelstrup, G. Hommes, T. G. Pierce, and V. V. Bulatov. Enabling strain hardening simulations with dislocation dynamics. *Modelling Simul. Mater. Sci. Eng.*, 15:553–595, 2007.

[2] V. Bulatov, W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, K. Yates, and T. Arsenlis. Scalable line dynamics in `paradis`. *Conference on High Performance Networking and Computing, Proceedings of the Proceedings of the ACM/IEEE Super Computing 2004 Conference (SC'04), p. 19.*, page 19, 2004.

[3] R.B. Sills and W. Cai. Efficient time integration in dislocation dynamics. *Modelling Simul. Mater. Sci. Eng.*, 22:025003, 2014.

[4] M Verdier, M Fivel, and In Groma. Mesoscopic scale simulation of dislocation dynamics in fcc metals: Principles and applications. *Modelling and Simulation in Materials Science and Engineering*, 6(6):755, 1998.

[5] B Devincre, R Madec, G Monnet, S Queyreau, R Gatti, and L Kubin. Modeling crystal plasticity with dislocation dynamics simulations: The micromegas code. *Mechanics of Nano-objects. Presses de l'Ecole des Mines de Paris, Paris*, pages 81–100, 2011.

[6] KW Schwarz. Simulation of dislocations on the mesoscopic scale. i. methods and examples. *Journal of Applied Physics*, 85(1):108–119, 1999.

[7] N Ghoniem, M, S-H Tong, and LZ Sun. Parametric dislocation dynamics: a thermodynamics-based approach to investigations of mesoscopic plastic deformation. *Physical Review B*, 61(2):913, 2000.

[8] Jianming Huang and Nasr M Ghoniem. Accuracy and convergence of parametric dislocation dynamics. *Modelling Simul. Mater. Sci. Eng*, 10:1–19, 2002.

[9] J. D. Lambert. *Numerical Methods for Ordinary Differential Systems.* John Wiley & Sons, Ltd., West Sussex, England, 1991.

[10] U. M. Ascher and L. R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations.* SIAM, Philadelphia, 1998.

[11] C. T. Kelley. *Iterative Methods for Linear and Nonlinear Equations.* SIAM, Philadelphia, 1995.

[12] D. G. Anderson. Iterative procedures for nonlinear integral equations. *J. Assoc. Comput. Machinery*, 12:547–560, 1965.

[13] N. N. Carlson and K. Miller. Design and application of a gradient-weighted moving finite element code I: in one dimension. *SIAM J. Sci. Comput.*, 19(3):728–765, 1998.

[14] P. A. Lott, H. F. Walker, C. S. Woodward, and U. M. Yang. An accelerated Picard method for nonlinear systems related to variably saturated flow. *Advances in Water Resources*, 38:92–101, 2012.

[15] H. F. Walker, C. S. Woodward, and U. M. Yang. An accelerated fixed-point iteration for solution of variably saturated flow. In J. Carrera, editor, *XVIII International Conference on Water Resources*, 2010. XVIII International Conference on Computational Methods in Water Resources (CMWR 2010), Barcelona, Spain, June 21-24, 2010.

[16] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.

[17] W. Cai and V. Bulatov. Mobility laws in dislocation dynamics simulations. *Materials Science and Engineering A*, 387:277, 2004.

[18] N. R. Barton, J. V. Bernier, R. Becker, A. Arsenlis, R. Cavallo, J. Marian, M. Rhee, H.-S. Park, B. A. Remington, and R. T. Olson. A multiscale strength model for extreme loading conditions. *J. Appl. Phys.*, 109:073501, 2011.

[19] D. Kincaid and W. Cheney. *Numerical Analysis.* AMS, Providence, 2002.

[20] C.A. Kennedy and M.H. Carpenter. Additive Runge-Kutta schemes for convection-diffusion-reaction equations. *Appl. Numer. Math.*, 44:139–181, 2003.

[21] G. Söderlind. The automatic control of numerical integration. *CWI Quarterly*, 11:55–74, 1998.

[22] G. Söderlind. Digital filters in adaptive time-stepping. *ACM Trans. Math. Soft.*, 29:1–26, 2003.

[23] G. Söderlind. Time-step selection algorithms: Adaptivity, control and signal processing. *Appl. Numer. Math.*, 56:488–502, 2006.

[24] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers. *ACM Trans. Math. Soft.*, 31(3):363–396, 2005.

[25] S.D. Cohen and A.C. Hindmarsh. CVODE, a stiff/nonstiff ODE solver in C. *Computers in Physics*, 10(2):138–143, 1996.

[26] S.R. Billington. Type-insensitive codes for the solution of stiff and nonstiff systems of ordinary differential equations. Master's thesis, University of Manchester, United Kingdom, 1983.

[27] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II – Stiff and Differential-Algebraic Problems*. Springer, Heidelberg, 2010.

[28] A. Kværnø. Singly diagonally implicit Runge-Kutta methods with an explicit first stage. *BIT Numerical Mathematics*, 44:489–502, 2004.

[29] H. F. Walker and P. Ni. Anderson acceleration for fixed-point iterations. *SIAM J. Numer. Anal.*, 49:1715–1735, 2011.

[30] H. Fang and Y. Saad. Two classes of multisecant methods for nonlinear acceleration. *Numer. Linear Algebra Appl.*, 16:197–221, 2009.

[31] A. Toth and C. T. Kelley. Convergence analysis for Anderson acceleration. Submitted, 2013.

[32] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19:400–408, 1982.

[33] S. C. Eisenstat and H. F. Walker. Choosing the forcing terms in an inexact Newton method. *SIAM J. Sci. Comput.*, 17:16–32, 1996.

[34] Y Saad and M H Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput*, 7(3):856–869, 1986.

[35] C.S. Woodward et al. SUNDIALS. http://computation.llnl.gov/casc/sundials/main.html.

[36] P.N. Brown and Y. Saad. Hybrid krylov methods for nonlinear systems of equations. *SIAM J. Sci. Stat. Comp.*, 11:450–481, 1990.

[37] D.R. Reynolds et al. The ARKode solver. http://faculty.smu.edu/reynolds/ARKode/.

Acknowledgments

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy Office of Advanced Scientific Computing Research and the National Nuclear Security Agency. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC. This work was also performed by Southern Methodist University under subcontract B603971 from Lawrence Livermore National Laboratory. Release number LLNL-JRNL-656343.